AFRL-RI-RS-TR-2010-192

AN EXTENSIBLE MODEL AND ANALYSIS FRAMEWORK

University of California, Berkeley

November 2010

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2010-192 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

BRIAN C. ROMANO
Work Unit Manager

/s/

MICHAEL WESSING, Acting Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
NOVEMBER 2010	Final Technical Report	February 2008 – August 2010

4. TITLE AND SUBTITLE

AN EXTENSIBLE MODEL AND ANALYSIS FRAMEWORK

5a. CONTRACT NUMBER
FA8750-08-2-0001

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Christopher Brooks, Edward A. Lee

5d. PROJECT NUMBER
558E

5e. TASK NUMBER
CP

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California
Berkeley
337B Cory Hall
Berkeley CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIEF
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RIEF

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2010-192

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT The goal of this project was to build on top of a pre-existing, open-source modeling and analysis framework known as Ptolemy II (http://ptolemy.org). The University of California, Berkeley worked with the Air Force Research Laboratory, Rome Research Site on adapting Ptolemy II for modeling and simulation of large scale dynamics of Political, Military, Economic, Social, Information and Infrastructure (PMESII) on the National Operational Environment Modeling (NOEM) project. Key characteristics of such systems include complex, multi resolution continuous and discrete dynamics and large parameter sets. The following specific capabilities were prototyped in Ptolemy II and delivered via version control and software releases. Each of these capabilities specifically supports one or more mechanisms for building and maintaining large models: 1. A Publisher/Subscriber mechanism that enables communication between disparate parts of a large model. 2. Elimination of quadratic complexity algorithms in running Ptolemy II models. 3. Width inference, which facilitates parameterized width for communication channels. 4. Lazy composite actors, which make it faster to open and browse large models. 5. Modular code generation, which compiles models into a form that can execute considerably faster than the default interpreted execution.

15. SUBJECT TERMS
TANGRAM, National Operational Environment Modeling (NOEM), Political, Military, Economic, Social, Information and Infrastructure (PMESII), Modeling, Ptolemy

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			BRIAN C. ROMANO
U	U	U	UU	26	19b. TELEPHONE NUMBER (*Include area code*) N/A

Abstract

The goal of this project was to build on top of a pre-existing, open-source modeling and analysis framework known as Ptolemy II (http://ptolemy.org). The University of California, Berkeley worked with the Air Force Research Laboratory, Rome Research Site on adapting Ptolemy II for modeling and simulation of large scale dynamics of Political, Military, Economic, Social, Information and Infrastructure (PMESII) on the National Operational Environment Modeling (NOEM) project. Key characteristics of such systems include complex, multiresolution continuous and discrete dynamics and large parameter sets. The following specific capabilities were prototyped in Ptolemy II and delivered via version control and software releases. Each of these capabilities specifically supports one or more mechanisms for building and maintaining large models:

1. A Publisher/Subscriber mechanism that enables communication between disparate parts of a large model.
2. Elimination of quadratic complexity algorithms in running Ptolemy II models.
3. Width inference, which facilitates parameterized width for communication channels.
4. Lazy composite actors, which make it faster to open and browse large models.
5. Modular code generation, which compiles models into a form that can execute considerably faster than the default interpreted execution.

Table of Contents

1 Participants

PRINCIPAL INVESTIGATOR:
EDWARD A. LEE
FUNDED TECHNICAL STAFF:
CHRISTOPHER BROOKS
MARY STEWART
BERT RODIERS -JR. DEV. ENGINEER
MAN-KIT (JACKIE) LEUNG -JR. DEV. ENGINEER
UNFUNDED TECHNICAL STAFF:
STAVROS TRIPAKIS
UNFUNDED GRADUATE STUDENTS:
DAI BUI
BEN LICKLY
BUSINESS ADMINISTRATOR:
GLADYS KHOURY
TRACEY RICHARDS

2 Summary

This is the final report of the Extensible Modeling and Analysis Framework (EMAF) project. The Extensible Modeling and Analysis Framework is collaboration between Air Force Research Laboratory, Rome Research Site and the Ptolemy project. This 30-month project started on February 29, 2008 and was scheduled to end on June 30, 2010. Because of an interruption in funding, the project was extended until August 28, 2010.
The objective of this effort was to build on top of a pre-existing, open-source modeling and analysis framework known as Ptolemy II [1] (http://ptolemy.org), adapting the framework for the rapid construction and configuration of modeling and analysis systems that incorporate disparate technologies. The purpose of this gap-filling project was to develop technologies for future incorporation into large-scale modeling and analysis systems, with specific focuses on scalable algorithm description, composition of heterogeneous components, and synthesis of efficient deployable decision-support systems that exploit multicore and distributed computing platforms. Ptolemy II was adapted for modeling and simulation of large scale dynamics of Political, Military, Economic, Social, Information and Infrastructure (PMESII) on the National Operational Environment Modeling (NOEM) project

2.1 Ptolemy capabilities

The following specific capabilities have been prototyped in Ptolemy II and delivered via our Subversion (SVN) version control system repository in open-source form and via software releases under a BSD-style license. Each of these capabilities specifically supports one or more mechanisms for building and maintaining large models:

1. A Publisher/Subscriber mechanism that enables communication between disparate parts of a large model. This was a true collaborative effort with AFRL.
2. Elimination of quadratic complexity algorithms in running Ptolemy II models.
3. Width inference, which facilitates parameterized width for communication channels (prompted by a query from AFRL).
4. Lazy composite actors, which make it faster to open and browse large models.
5. Modular code generation, which compiles models into a form that can execute considerably faster than the default interpreted execution.

3 Introduction

This project was a highly targeted effort to extend Ptolemy II with modularity, scalability, and code generation mechanisms, and to harvest, adapt, and retarget open source extensions to Ptolemy II that have been made in the context of the Kepler project (http://kepler-project.org) and in other funded Ptolemy II-related efforts. The Kepler project has built on Ptolemy II a grid-enabled scientific workflow development environment that shares many of the objectives of Tangram, but with an entirely different application focus (analysis of scientific data rather than intelligence data). Other projects have supported translation of Ptolemy II models into efficient standalone C programs. Our focus was to adapt and extend these technologies to support rapid design, deployment, and reconfiguration of large-scale analysis algorithms.

4 Methods, Assumptions, and Procedures

This work built on the pre-existing Ptolemy II design environment, which is a set of Java packages supporting heterogeneous, concurrent modeling, simulation, and design of component-based systems. The emphasis is on a clean, modular software architecture, divided into a set of coherent, comprehensible packages. The kernel package supports definition and manipulation of clustered hierarchical graphs, which are collections of *entities* and *relations* between those entities. The actor package extends the kernel so that entities have functionality and can communicate via the relations. The *domains* extend the actor package by imposing *models of computation* on the interaction between entities. The Ptolemy II graphical user interface is called Vergil. Vergil itself is a component assembly defined in Ptolemy II. Ptolemy II includes facilities for code generation from models. The latest release of Ptolemy II consists of 640K lines of Java code, of which 340K lines are non-commented, non-blank Java code.

5 Results and Discussion

5.1 Events and Deliverables

Table 1: Rome/Berkeley Events and Deliverables

Date	**Event**
2/08	Extensible Modeling and Analysis Framework (EMAF) project starts
5/08	Kickoff meeting, Berkeley
12/08	NOEM day, Rome
4/07	Ptolemy II 7.0.1 shipped
4/09	Extensible Modeling and Analysis Framework (EMAF) Program Review at Berkeley
4/09	Eighth Biennial Ptolemy Miniconference and Ptutorial held in Berkeley. AFRL and ITT project staff attend
10/09 through 2/12/10	Hold on funding
11/09	NOEM day, Rome
3/10	Ptolemy II 8.0.beta shipped
2/28/10	Ptolemy Technical Exchange, Rome
6/30/10	Original end date
8/28/10	New end date

5.2 Project Results as Compared with the SOW

Below we quote from section 4 of the statement of work and update the status of each item.

> "4.1 The contractor shall collaboratively develop enhancements, adapt, and incrementally improve performance, extensibility, integration, and usability a pre-existing systems dynamics modeling framework building on an open-source, extensible, open architecture to meet and improve the modeling needs of AFRL/RI researchers."

1. Under a previous contract, AFRL and Berkeley jointly developed the Publisher, Subscriber and SubscriptionAggregator actors. These actors pass data across levels of hierarchy. The AFRL NOEM model uses Publisher actors to publish data from deep within the model to upper levels where the data is read by the Subscriber or SubscriptionAggregator actors.

In Figure 1, note that SubscriptionAggregator will receive the values of both Publisher and Publisher2 because the channel of the SubscriptionAggregator is a regular expression that matches the channel names of both Publishers. SubscriptionAggregator can either add or multiply the values that are read. Note that Publishers and Subscribers or SubscriptionAggregators need not be on the same level of hierarchy as is illustrated in this example.

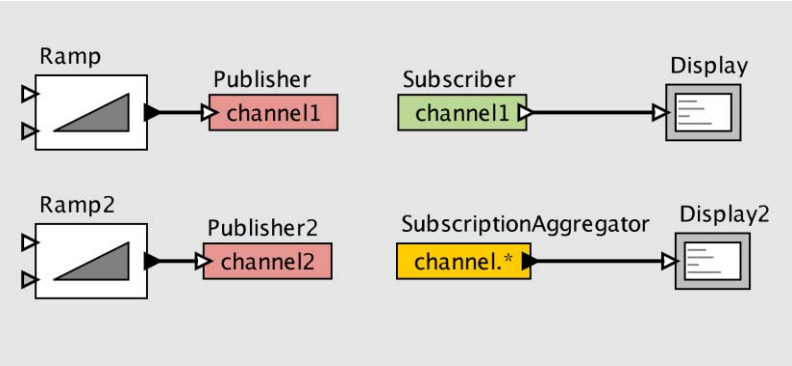

Figure 1: *Uses of Publisher, Subscriber and SubscriptionAggregator*.

Under this contract, we improved performance of these actors, solved a number of bugs involving actor oriented classes and refactored the implementation so that TypedCompositeActors are now aware of any Publisher/Subscriber or SubscriptionAggregators contained by the TypedCompositeActor. This refactoring was necessary in part to support modular code generation.

2. To enhance scalability, Ptolemy II supports connections with parameterized width. In Figure 2, the top-level composite actor has three Const actors that are connected to the port of a CompositeActor that contains a CompositeActor that in turn contains an AddSubtract actor. Jason E. Smith (ITT) pointed out that only the first Const actor was fired because the width of the relations defaulted to 1. In theory, setting the widths of the

two inner relations to 0 should cause the proper width to propagate up to the outer level and the width should be set to 3, which would cause all three Const actors to fire and their values read.

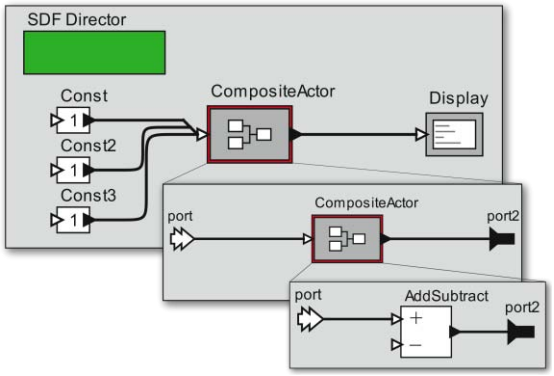

Figure 2: A model with different widths.

The strategy that we had initially implemented turned out to create awkwardnesses for certain kinds of models. Bert Rodiers identified an algorithm that could infer widths for a broader class of examples, and showed that no solution could handle all possible cases without manual annotation.

In particular his solution addressed the problem that Multiport composites cannot properly infer their width. Designing and implementing the width inference algorithm was a significant amount of effort, see Bert's presentation from the 04/09 program review and the technical report [2].

3. Many bug fixes and performance enhancements were done in conjunction with the ITT staff. See the quarterly reports for details, below are the highlights:

- Fixed a problem involving incrementing the workspace version when creating receivers, tableaus and effigies.
- The plotter can now plot large datasets efficiently.
- The plotter now optionally fills every few seconds, which improves performance.
- The Publisher.attributeChanged() method was improved so that the model opens 40% faster. This and other changes decreased opening time from 50 seconds to 25 seconds on a particular large AFRL model, though the run time ended up taking 25 seconds longer. This type of change is worth it, as users can open the model, make changes and then run the model.
- CompositeActors may be lazily evaluated, which makes the model open more quickly. See below for details.
- Refactoring of Publisher channel names is now better supported. If the propagateNameChanges parameter of a Publisher is true and the channel name of a Publisher is updated, then matching Subscribers channel names will be updated.
- The error dialog box and stack trace dialog box have been updated to include a "Go To Actor" button that will open the container of the object that caused the error.

4. Folded in an automatic graph layout tool developed by University of Kiel (at no cost to this contract). This tool allows layout of complex models. (6/09 status report)

Lazy Evaluation

When a large hierarchical model is opened, it is not always necessary to completely expand the model and instantiate each actor in the model. For example, the user might only be interested in a certain portion of the model; there is no need to instantiate all of the actors in the model until the user wants to view that portion of the model or wants to run the entire model. To provide this functionality, we developed the LazyTypedCompositeActor class, which is an aggregation of typed actors with lazy instantiation. This is similar in spirit to lazy evaluation, a common feature of functional programming languages [3][4]. Our mechanism simply loads the XML text into memory without parsing it, deferring the parsing and instantiation of objects until there is demand for those objects. We also developed a "convertToLazy" tool that converts models to use LazyTypedCompositeActor. The way convertToLazy works is that you set a threshold (the default is 100), and any composite actor that contains more than the threshold number of actors inside is converted to lazy.

Table 2 shows the results for 5 runs of two models, a non-lazy version and a lazy version of a NOEM model from AFRL called Integration 433. In the non-lazy version, for 5 runs, the opening time took (on average) 68 seconds and the run time took 356 seconds, for a total of 424 seconds. In the lazy version, for 5 runs, the opening time took 15 seconds (a 4.5x speed up), but the execution time took 387 seconds for a total of 402 seconds. This indicates that Lazy evaluation is faster to open, takes longer to run (the first time), but the total opening and run time is about 5% faster. In addition, the Lazy version consumes about 4% less memory.

Table 2: Lazy Evaluation of the Integration 433 model

Model Type	**Opening Time**	**First Run**	**Opening + First Run**	**Memory**
Non-lazy	68 sec.	356 sec.	424 sec.	847393 K
Lazy	15 sec.	387 sec.	402 sec.	814687 K

5.2.1 Higher Order Components

> "4.1.1 Construction of a library of higher-order components supporting scalable data analysis using design patterns such as MapReduce."

On this effort, the project had a change of strategy: instead of focusing on higher-order components, the project focused on model transformations. (12/08 presentation) A first version of Thomas Feng's Graph Transformation facility shipped in 4/08 [5] (12/08 presentation). This facility is based on triple graph grammar, which includes recognizing common design patterns in the models in a static analysis, replacing existing design patterns with more efficient ones, and reusing design patterns by incorporating them into new models.

In addition, the Ptolemy Hierarchical Orthogonal Multi-Attribute Solver (PtHOMAS) project (with additional funding from the Bosch Research Center, Palo Alto), demonstrated a system for associating properties from a domain-specific ontology with elements of a model (such as ports and parameters), and performing inference and compatibility checking to ensure correct composition of models. In one application,

whether a signal on a port is constant over time or non-constant is inferred from the structure of the model, and model transformation can then be used to optimize the model, sometimes considerably reducing its size. This work leveraged the Ptolemy II type system, which is known to scale well to large models.
We implemented a first draft design and are currently refactoring the design to make the system easier to use The ontology work is directly applicable to NOEM because ontologies help make models more correct by detecting errors in mismatched inputs and outputs such as adding numbers that are different currencies or misinterpreting a graduation rate as a dropout rate. (6/09 status report and subsequent reports)

5.2.2 Code Generation

> "4.1.2 Development of code generation capabilities that synthesize deployable standalone analysis and decision support applications from Ptolemy II models."

1. To improve the runtime performance of the NOEM model, we used code generation to create a faster version of the model. Our first effort was to generate C code. In January 2008, we started with the March, 2007 NOEM model, which took 217 seconds to run in interpreted mode. After code generation, our C version took 3 seconds to run. However, drawbacks of this method included:
(a) Code needed to be generated each time the model changed. Code generation took about 400 seconds, the C compiler took 140 seconds and the run took 3 seconds for a total of 543 seconds. For comparison purposes, in interpreted mode, opening the model took 224 seconds and running the model took 217 seconds for a total of 431 seconds. Thus, the end-to-end solution indicates that C code generation is slower during development of the model, assuming that the model changes between runs. Nevertheless, this technique could be useful for creating deployable models for use in the field, where changes to the model will be infrequent.
(b) Using C requires that end users have a C compiler installed and that the Java Native Interface (JNI) be used to read data back in from the code generated executable. It was felt that both of these issues would made C code generation less desirable.

Our next step was to generate Java code, which would avoid point (b) above. In February of 2009, we were able to generate Java code for the June 2008 NOEM model. Unfortunately, we cannot compile the resulting Java code because there is a 64k byte limit on the size of the byte code for a method and there are limits to the number of variables in a Java class. In the generated code, we currently have 14k variables and 27k methods, which means it is unlikely that all the byte code will fit within one class.
In addition, this solution did not solve the problem of needing to regenerate code each time the model changed, which we address below.

2. We developed Modular Code Generation so that we can get the major performance improvements needed by this project. The idea is that we will generate one Java class per group of composite actors. This could solve both problems above because the generated code is Java and, in certain circumstances at least, code may need to be regenerated for only part of the model when that part of the model changes. In Figure 3, the top-level model contains a ModularCodeGenTypedCompositeActor, which is lazy, meaning that

the inner composites will not be opened until necessary. The inner composites both have SDFDirectors; we refer to these as Opaque Composites.

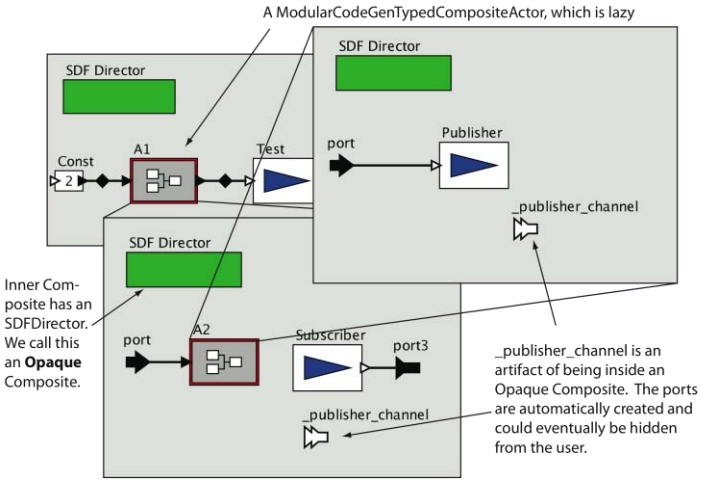

Figure 3: *An example of Modular Code Generation.*

The complete solution to determining what parts of the model need to be regenerated is an open research question. However, as part of future work, it might be possible to mark parameters as being non-constant so that model developers would not need to regenerate the entire model. For the model users, when doing a parameter sweep, it should be fairly easy to have parameters change and not regenerate code. In Ptolemy Classic, we were able to mark certain parameters and non-constant and then use this in the code generator so that these parameters would be read at run time of the generated code.

An example is shown in Figure 3, where the model has two key properties:

(a) It is lazy. The inside of this model is not expanded until it is needed.

(b) When it fires the first time, it will generate code for its inside model. Subsequently, it will use that code and not look at the inside model. If the model is changed (anywhere) then it will regenerate code. If you close and re-open the model and run it, it will run the generated code and will never expand the inside model.

This work turned out to have some deep implications, and open questions remain (discussed further below). Stavros Tripakis, Dai Bui, Bert Rodiers, Jorn Janneck and Edward A. Lee developed a paper "Compositionality in Synchronous Data Flow: Modular Code Generation from Hierarchical SDF Graphs" [6], that describes this work, the abstract of which is below:

> "Hierarchical SDF models are not compositional: a composite SDF actor cannot be represented as an atomic SDF actor without loss of information that can lead to deadlocks. Motivated by the need for modular code generation from hierarchical SDF models, we introduce in this paper DSSF profiles. This model forms a compositional abstraction of composite actors that can be used for modular compilation. We provide algorithms for automatic synthesis of non-monolithic DSSF profiles of composite actors given DSSF profiles of their sub-actors. We show how different tradeoffs can be explored when synthesizing such profiles, in terms of modularity (keeping the size of the generated DSSF profile minimal) versus reusability (preserving information necessary to avoid deadlocks) as well as algorithmic complexity. We show that our method guarantees maximal reusability and report on a prototype implementation."

The code generator is a template driven, where actors have templates that define the functionality in an intermediate language. Thus, each actor must have a template or adapter. We extended the code generator by creating wrappers that can automatically call the compiled code of a predefined custom actor, obviating the need for a manually designed template. This is critical for the NOEM project in that the code generator may now call NOEM custom actors such as the Economics actor without requiring hand generation of an adapter. The downside to this approach is that the Economics actor is wrapped in a composite actor that has a director, and that data needs to be translated back and forth from raw Java types to Ptolemy token types. This work is substantially complete, though further testing of multirate models is warranted.

3. Bert Rodiers reimplemented Publisher/Subscriber so that they are now properties of ports rather than actors. This work was required for modular code generation. In addition, we modified Publisher and Subscriber so that both actors now have a flag named "global" that specifies whether the published data is global. If a Publisher has this flag set to true, then a subscriber anywhere in the model that references the same channel by name will see values published by the Publisher. If the value is false (the default), then only those Subscribers that are fired by the same director will see values published on this channel.

To test modular code generation, we created scripts that generate models with various levels of hierarchy and numbers of actors. We found a particular model, "hierarchicalModel 3 6" (Figure 4) that roughly matched the size of the NOEM Integration 433 model. In Figure 4, the top level has three composites that each contain three composites that eventually contain 25 scale actors.

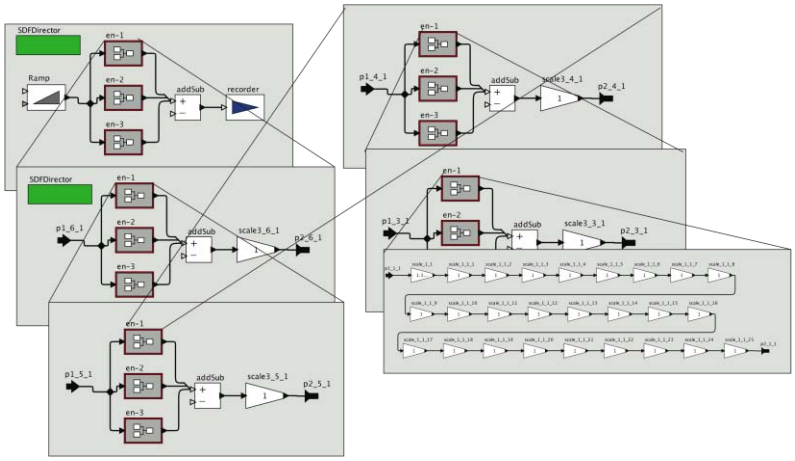

Figure 4: *A test model that contains 19683 entities.*

A comparison of the key model complexity metrics may be found in Table 3.

Table 3: Comparison of the model structure

Metric	**Integration 433**	**hierarchicalModel 3 6**
AtomicEntities	13384	19683
CompositeEntities	1639	1092
Depth of Hierarchy	7	6

For each test, we start with the same basic model of 19683 actors, but change the type of the composite from TypedComposite to LazyTypeComposite and then to ModularCodeGenTypedComposite. We also change whether the inner levels have an SDFDirector. Levels that have an SDFDirector are known as Opaque composites, levels that do not have an SDFDirector are known as Transparent composites. The results of these tests are in Table 4.

Table 4: Sample data from a model with 19317 Entities over 365 iterations

#	Composite	Opaque or Transparent	First Run	Average of subsequent runs
1.	TypedComposite	Transparent	42 sec.	34.3 sec.
2.	TypedComposite	Opaque	40.9 sec.	35.8 sec.
3.	LazyTypedComposite	Transparent	60.6 sec.	34.4 sec.
4.	LazyTypedComposite	Opaque	59.8 sec.	35.1 sec.
5.	ModularCodeGen	Opaque	384 sec. compile, 3.5 sec. run	3.5 sec.
6.	TypedComposite Codegen	Transparent	360 sec. compile, 0.3 sec. run	0.3 sec.

Line 1 is the standard model run in interpreted mode.
Line 4 is a Lazy model that will open quickly. Model developers would find this useful.
Line 5 is the key point. The first run includes code generation time. After we use Modular Code Generation, the run time is about 10% of the time (3.5 sec. vs. 34.3) when compared to regular interpreted mode. Model developers would find this very useful. Ideally, code generation only occurs for modules that are changed, but this is only partially implemented because width inference and type inference can have global effects on a model.
Line 6 generates code for the entire model. Code generation for the entire model occurs each time the model is changed. This would be good for deployment. Notice that the execution time of the model is about 1% of that in interpreted mode. This is a substantial speedup.

Note that we use Opaque composites in Modular Code Generation for the following reasons:

1. Adding Directors to Composites starts moving the model towards being more parallelized.
2. Using LazyTypedCompositeActor means the model opens more quickly than with TypedAtomic.
3. Intellectual Property is more easily encapsulated in an Opaque actor.
4. A straightforward clustering of a transparent actor into an Opaque actor means that its execution must occur in a single fire() method. Such clustering cannot be done in general without changing the semantics of the model because of the lack of modularity of the SDF model of computation. It is hard to know when such clustering can be done without changing the semantics of the model because with clustering, feedback loops may cause deadlock.

To address the last issue, there are three possibilities. One is to train the builders of models to design their models with Opaque composite actors. A second approach is to automatically create clusters that do not change the semantics of the model. This is the approach used in Joe Buck's Ph.D. thesis [7], which includes information about clustering. A third approach is to break apart the single fire() method of an Opaque composite actor. This is an ongoing research problem on which we are making progress, but the work is not complete. Our research on such modular code generation shows that Opaque vs. Transparent is central to incremental compilation.

Limitations in our current Modular Code Generation System:

1. The test model above does not use Publisher/Subscriber. Modular Code Generation does support Publisher/Subscriber, but we found performance unsatisfactory when a model has a large number of Publishers and Subscribers. Performance analysis using a tool like JProbe could quickly suggest a solution.
2. The NOEM model uses feedback loops, which make clustering, and using Modular Code Generation difficult. Solving this problem automatically, without requiring designers to modify the model, is a key part of a proposal for follow-up work that is under consideration.
3. It is difficult to know when to regenerate code for a composite. For example, if a parameter changes, does it force regeneration of an inner composite? Solving this problem could be part of future work.
4. The template language is rather crude. It should be revamped to use openArchitectureWare (oAW) which is now part of the Eclipse Modeling Framework (EMF). Improving the template language is a key part of a proposal for follow-up work that is under consideration.

5.2.2.1 Multicore Machines

> "4.1.2.1 Adaptation of code generation techniques to automatically configure for multicore and networked compute platforms."

We rearchitected our code generation framework to aid in generating code with different execution semantics. In particular, we moved Synchronous Data Flow (SDF) specific code out of the code generation kernel. This makes it easier to use the Process Networks director, which executes submodels in separate threads, transparently exploiting multicore machines (12/09 status report).

5.2.3 Scalability Metrics

"4.1.3 Design of scalability metrics that can be used to assess the degree to which models scale to match data set sizes."

1. We used Thomas Feng's Graph Transformation work to build large models. (12/08 presentation)
2. We generate more statistics at run time. These statistics include the time and memory consumed for the various phases of execution (opening time, preinitialize(), initialize(), prefire(), fire() and postfire()). (12/08 presentation)
3. We asked for guidance at our 12/08 meeting and determined that we should focus on performance improvements. (12/08 presentation)
4. We used scripts to generate large models for performance analysis. (12/08 presentation)

"4.1.3.1 Application of the scalability metrics to scalability improvements."

1. We used automatically generated large models for performance analysis and improvements.
2. In collaboration with ITT, we implemented many performance improvements, especially involving start-up time. (slide 60 of the 12/08 presentation)
3. When a large hierarchical model is opened, it is not always necessary to completely expand the model and instantiate each actor in the model. For example, the user might only be interested in a certain portion of the model; there is no need to instantiate all of the actors in the model until the user wants to view that portion of the model or wants to run the entire model. To provide this functionality we developed the LazyTypedCompositeActor class, explained above (8/09 status report).

5.2.4 Configuration Files

"4.1.4 Editing tools for Ptolemy II configuration files to support construction of more specialized modeling, analysis, and decision-support tools."

"4.1.4.1 Synthesis of Ptolemy II configuration files (which are currently in XML) from abstracted specifications."

1. Our proposal is to use Eclipse or Netbeans Rich Client Platform (RCP). We call this the Triquetrum Project. Configuration files support narrower variability than Triquetrum/RCP. Triquetrum/RCP supports assembling in arbitrary ways. (12/08 presentation)
2. Prototyped OSGi component architecture for use with Netbeans and Eclipse. (12/08 presentation)
3. Prototyped embedding vergil viewer in Eclipse. (12/08 presentation)
4. Partitioned Ptolemy into groups of components and refactored code to remove spurious dependencies.
5. Provided feedback on Kepler's build configuration system. (2/10 report)
6. Collaborated with developers from University of Kiel and learning about the Eclipse Modeling Framework (EMF) and metamodeling to determine if we can leverage their work and provide a new user interface.

5.2.5 Dependency Management

"4.1.5 Tools for managing dependencies between modeling and analysis tools and component libraries that are used. Integration of such tools with Ptolemy II configuration management."

1. New Associative Object Model of Integration (NAOMI) project, a separate project funded by Lockheed Martin ATL. NAOMI is a tool that enables disparate modeling systems to work together where tools publish attributes and subscribe to the attributes of published by other tools. This is much safer than syntactic transformations that do not take into account semantics. This project has ended. (12/08 presentation)

5.2.6 Parallel and Distributed Execution

"4.1.6 Tools for managing parallel and distributed execution of analysis models."

1. Jackie Leung prototyped a C code generation facility that uses Kahn Process networks, which uses one thread per actor. (12/08 presentation)
2. Developed Threaded Composite actor, which allows easy creation of multi-threaded models in non-multithreaded domains. However, models that use Publisher/Subscriber would require a transformation [8] (12/08 presentation)
3. We are collaborating with Kepler, which includes ways to execute Ptolemy models using Grid computing. (12/08 presentation)

5.2.7 Format of Deliverables

"4.1.7 Deliver all computer software developed or assembled to be completely maintainable and modifiable without reliance on any non-delivered computer programs or documentation in accordance with the Contract schedule and the following. Deliver software acquired and included as a component in the software developed and delivered.

We are following academic best practices including nightly builds and code reviews.

"4.1.7.1 All information technology items must be Year 2000 compliant or non-compliant items must be upgraded at no additional cost to be Year 2000 compliant. Year 2000 compliant means information technology that accurately processes data/time data (including, but not limited to calculating, comparing, and sequencing) from, into, and between the twentieth and twenty-first centuries, and the years 1999 and 2000 and leap year calculations. Furthermore, Year 2000 compliant information technology, when used in combination with other information technology, shall accurately process date/time data if the other information technology properly exchanges date/time data within it."

Ptolemy II uses Java, which is Y2K compliant, and Ptolemy II is itself Y2k compliant.

"4.1.7.2 Deliver all computer software developed under this effort as source and object (executable) code. Include the commented source listings and source coded for the target computer system. (CDRL A003)"

Our development tree is available via the Subversion (SVN) version control system at http://chess.eecs.berkeley.edu/ptexernal

"4.1.7.2.1 Deliverables must not inhibit re-use or redistribution. AFRL must be able to use the results of this effort (software and concepts) to develop derivative products in the future without additional licensing costs or distribution restrictions. Therefore, deliverables must not require any dependencies that extend, adapt, or introduce additional licensing or redistribution terms."

Ptolemy II is released under the BSD license.

> "4.1.7.2.2 Java technologies that are freely available, freely re-distributable, and platform independent, must be used to develop any of the deliverables that provide functionality specified by the contract requirements."
> Ptolemy II uses Java open source technologies whenever possible, all parts of Ptolemy necessary for this project are freely available, freely re-distributable and platform independent.
> "4.1.7.3 Develop complete software documentation to include installation, user, and maintenance instructions as appropriate for the application. (CDRL A004)"

The Ptolemy Design doc was shipped in April 2008. We are converting the design doc sources from a proprietary format (Adobe Framemaker) to an open source format (L^AT_EX).

6 Conclusions

During this project, we further refined Publisher/Subscriber, improved performance, improved width inference, added lazy composite actors and developed modular code generation. All of these features improve the experience of the model builder and improve the performance of the simulation for the end user.

7 Recommendations

We recommend that we continue work on modular code generation, see the appendix for our proposed follow on work.

7.1 Lessons Learned

Below is a list of issues that have come up that affected this contract.

1. The level crossing links used by the Publisher and Subscriber actors drastically break modularity of a model. Level crossing links are much like global variables and goto statements in procedural languages. There are two key problems:

(a) In order to generate code for a submodel, that submodel has to be able to execute as a unit. In Ptolemy, this (currently) means it has to have its own director, although we have a research effort towards a more flexible notion of separately compilable submodel [6]. We have successfully augmented Publisher and Subscriber so that, with the optional "global" flag set, communication can occur across domains controlled by different directors. The semantics is clean, and the mechanism is supported in simulation.

(b) To support the global flag in code generation, we have prototyped a mechanism that mirrors what we do in simulation. The current implementation, however, is very inefficient, and in a model with many Publisher and Subscriber actors, this mechanism eats away much of the speed improvement gained from code generation. I believe that we can improve this considerably for certain cases, but if we want to completely preserve modularity (a submodel never has to be regenerated if it does not change internally), then global communication between Publishers and Subscribers will be fundamentally expensive. It's easy to envision more efficient mechanism that do not preserve determinacy, but preserving determinacy, in our view, is an absolute requirement.

The next steps we would propose would be to:
(a) Tune the current mechanism to eliminate as much as possible the inefficiency in code generated for models that use Publisher and Subscriber, but not by going so far as to lose determinacy.
(b) Leverage the progress that Stavros Tripakis and Dai Bui have made on modular code generation to reduce or eliminate the requirement for a local director (by itself, this will not reduce the cost of Publisher and Subscribe, but it will make modular code generation usable for more models).
(c) Invent something better than Publisher and Subscriber that is efficient (e.g., if some delay in the communication is tolerable, then we can implement a very efficient and determinate mechanism).
See the appendix for technical details about our proposal for future work.

2. The last version of the NOEM model that we received was from June 2008. After that time, the model was ITAR restricted, and we are not equipped to safely handle ITAR-restricted data. Having the model be ITAR restricted means that we are working on an old model, so we rely on the ITT team to validate our enhancements.
3. From October 2009 until February 12, 2010, the UC Berkeley was not paid. During that time, we eventually could no longer bill to the contract. The contract was extended.
4. We were surprised at the last minute when we discovered a performance problem with Modular Code Generation and models that have many Publishers and Subscribers. Including more performance tests in our test harness would help avoid this in the future. However, performance tests are notoriously difficult to maintain in a regression test suite, as they require human intervention to evaluate the results. The execution time of a test may vary from day to day because of loading on the test machine. As a result, frequent human intervention is necessary for performance regression tests.

8 References

[1] Johan Eker, Jorn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity—the Ptolemy approach. Proceedings of the IEEE, 1(2):127–144, 2003.

[2] Bert Rodiers and Ben Lickly. Width inference documentation. Technical Report UCB/EECS-2010-120, EECS Department, University of California, Berkeley, Aug 2010.

[3] D. P. Friedman and D. S. Wise. CONS should not evaluate its arguments. In Third Int. Colloquium on Automata, Languages, and Programming. Edinburg University Press, 1976.

[4] J. H. Morris and P. Henderson. A lazy evaluator. In Conference on the Principles of Programming Languages (POPL). ACM, 1976.

[5] Thomas Huining Feng. Model Transformation with Hierarchical Discrete-Event Control. PhD thesis, EECS Department, University of California, Berkeley, May 2009.

[6] Stavros Tripakis, Dai Bui, Marc Geilen, Bert Rodiers, and Edward A. Lee. Compositionality in synchronous data flow: Modular code generation from hierarchical SDF graphs. Technical Report UCB/EECS-2010-52, EECS Department, University of California, Berkeley, May 2010.

[7] J. Buck. Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model. PhD thesis, University of California, Berkeley, 1993.

[8] Edward A. Lee. ThreadedComposite: A mechanism for building concurrent and parallel Ptolemy II models. Technical Report UCB/EECS-2008151, EECS Department, University of California, Berkeley, Dec 2008.

9 Appendix: Technical portion of Follow-on Proposal

Below is the technical portion of our proposal for BAA-10-03-RIKA, "Program: Reasoning, Comprehension, Perception and Anticipation in Multi-Domain Environments."

9.1 Technical Program Summary

The technical approach consists of two distinct efforts: 1) Use code generation to speed up execution; and 2) Improve the user interface to facilitate development and maintenance of large models. We believe that using modular code generation will provide the necessary increase in runtime performance. Improving the user interface will increase model developer productivity.

9.2 Technical Program

Ptolemy II models contain a potentially very large number of actors. The actors contain ports through which they communicate with other actors. A special class of actors, created in an earlier project to support the NOEM effort, enables communication across disparate parts of a model using named channels in a publish-and-subscribe pattern. Actors are hierarchical, in that an actor may be defined as a composition of other actors. Such an actor is called a composite actor. A model is a composite actor that is not contained by any other composite actor. A composite actor may contain a director, in which case it is called an opaque composite actor. The director defines the model of computation of the model; specifically, it provides the algorithms and procedures needed to execute the model.

AFRL NOEM models tend to use the synchronous dataflow (SDF) director. In the past, NOEM models have used only a single director in a model, but a number of advantages emerge from using more directors throughout the hierarchical. First, it enables the use of modal models, which are composite actors with multiple modes of operation. Second, some scheduling algorithms can be made more efficient when there are more directors. Third, a model is more modular in that submodels are more self-contained. The main goal of this project is to improve the scalability of Ptolemy II models. Specifically, NOEM models tend to use a very large number of actors. The project will develop a suite of techniques that enable efficient editing and execution of such large models. We have identified the following problems that limit the scalability of Ptolemy II models. The Report numbers correspond with reports on the Ptolemy issue tracking system at https://chess.eecs.berkeley.edu/bugzilla.

- Report 339 - *Code generation/Clustering - opaque composite actors need to be able to have multiple fire methods.* As explained in [6], Achieving modular code generation (MCG) on arbitrary hierarchical dataflow models is challenging and will require implementing multiple fire methods for each opaque composite actor.

  Specifically, hierarchical SDF models are not compositional: a composite SDF actor cannot be represented as an opaque composite actor without loss of information that can lead to deadlocks. Motivated by the need for incremental and modular code generation from hierarchical SDF models, we introduced in [6] so-called deterministic SDF with shared FIFOs (DSSF) profiles. DSSF profiles enable a compositional abstraction of opaque composite actors that can be used for MCG. We have developed algorithms for automatic synthesis of non-monolithic DSSF profiles of composite actors given DSSF profiles of their sub-actors. We have shown how different tradeoffs can be explored when synthesizing such profiles, in terms of modularity (keeping the size of the generated DSSF profile small) versus reusability (preserving information necessary to avoid deadlocks) as well as algorithmic complexity. We have shown that our method guarantees maximal reusability and report on a prototype implementation. In the proposed project, we will reduce these theoretical results to practice, enabling MCG on arbitrary SDF models with directors inserted at various points in the hierarchy.

- Report 338 - *Parallel execution on multicore machines.* Even with performance improvements promised by MCG, models will continue to grow, and enabling the effective use of multicore architectures will improve performance for large models. Now that crossing links (used by Publisher/Subscriber actors) can be used across the boundaries of opaque composite, we can use ThreadedComposite [8], where each composite may be a separate thread that may be run on a separate core. Now that level crossing links can be used across the boundaries of opaque composites, we can use of Kahn Process Networks, where each Opaque composite actor is a separate thread and run on a separate core. Ptolemy Classic, the predecessor for Ptolemy II, includes a number of algorithms for partitioning and scheduling dataflow models on parallel machines. These will provide a good starting point for this work.
- Report 333 - *Given a Subscriber, find all Publishers or vice versa.* Adding a search mechanism can greatly facilitate the development and maintenance of models with many interacting parts. We are currently mentoring a group of students from CMU that is developing infrastructure that would place models in an XML database and allow searches. The student project will be end in August 2010, but their output will likely require rework to be useful.
- Report 342 - Code generation should handle custom actors and actors that do not have adapters (the templates used for code generation) so that custom actors (not in a standard actor library) can be easily included in MCG.
- Report 337 - *Componentization of Ptolemy using OSGi.* OSGi is the component system used by the Eclipse Integrated Development Environment. In addition, Eclipse is likely to be our next GUI. Such componentization is expected to significantly reduce the memory footprint of a Ptolemy II process, which will help improve performance on large models.
- Report 336 - *Instantiating custom actors.* Ptolemy II has ways of instantiating actors and saving them in a user library. However, it needs to be easier for actor developers to deploy actors to model builders. A complete solution probably depends on a mechanism like the OSGi integration, which will enable a clean plug-in architecture.
- Report 332 - *Breadcrumb bar and/or model tree to facilitate navigation.* The goal here is to improve navigation of large models by providing a breadcrumb bar and/or a model tree browser.
- Report 344 - *Use openArchitectureWare templates for code generation.* As we increasingly rely on code generation to get reasonable performance, it will become more important to be able to easily develop and maintain the code generation actor library. Today, the actor library for code generation is given using a rather crude template file format that is inherited from the previous code generation package. We would like to replace this with a modern template code generation mechanism based on openArchitectureWare, which is now part of EMF (the Eclipse Modeling Framework). We have a rough plan for how to accomplish this. It is not easy, but it looks like it will work. This will make it much easier to write good code generation templates.

- *Support for Rome/AFRL programmers.* We propose to continue to work with AFRL personnel and NOEM project participants, as needed, to assist with design techniques and to help identify and correct bugs or limitations that present obstacles.
- *Ptolemy II 9.0 and 10.0 releases.* This work directly benefits the NOEM project by providing a clean and stable platform upon which NOEM is based.

10 List of Symbols, Abbreviations and Acronyms

DSSF - Determinstic Synchronous dataflow with Shared Fifos. Used in modular code generation (MCG).
EMF – The Eclipse Modeling Framework
MCG - Modular Code Generation, where one Java class is generated per group of composite actors.
NOEM - National Operational Environment Modeling Project.
oAW - Open Architecture Ware, a template language used by the Eclipse Modeling Framework code generator.
PMESII - Political, Military, Economic, Social, Information and Infrastructure
PtHOMAS - Ptolemy Hierarchical Orthogonal Multi-Attribute Solver. A system for associating properties from a domain-specific ontology with elements of a model and performing inference and compatibility checking.
SDF - Synchronous Dataflow, a model of computation in which the number of data samples produced or consumed by each node on each invocation is specified a priori.
SVN - Subversion, the version control system that allows Berkeley to share research results.